Compiling Object-Oriented Languages

Andrew P. Black

How are OO Languages Different?

- methods instead of procedures
- method request instead of procedure call
- “full upward funargs”
- inheritance & encapsulation
  ⇒ frequent method requests

Method Request

- Method request, aka message send, is not the same as procedure call

How are OO Languages Different?

- subtyping
  - types dictate interface, not implementation
    - not in all languages
  - code to be executed not known at time of request
Procedure Call

- Code to be executed is identified by name at call site
- Compiler’s job:

```
MatAdd(aMatrix, aNumber)
```

```
MatAdd(m, n)
```

```
foreach i in m do ...
```

```
SetAdd(aSet, aNumber)
```

```
SetAdd(s, n)
```

```
i := findEmptySlot(s);
insertAt(s, i, n);
```

Method Request

- Code to be executed depends on the receiver of the request

```
anObject.add(aNumber)
```

```
add(n)
```

```
foreach i in self do ...
```

```
remove
```

```
add
```

```
add(n)
```

```
i := self.findEmptySlot;
self.insertAt(i, n);
```

Implementing Objects

- Each object contains, conceptually:
  - a set of named methods
  - a set of named instance variables

```
x
y
```

```
myPoint
```

```
x
```

```
y
```

```
+ abs <
```

```
+ abs <
```

```
- x y
```

```
- x y
```

```
myPoint
```

```
myPoint
```

```
yourPoint
```

```
yourPoint
```

```
3
```

```
4
```

```
7
```

```
5
```

```
+ abs <
```

```
+ abs <
```

```
- x y
```

```
- x y
```
What does “send x” mean?

1. Find the representation of the receiver
2. Find its list of methods
3. Look for a method named “x”
4. If there is none, repeat above in the methods of the receiver’s superclass …

```
public class CartesianPoint implements Point{
    private double x; private double y;
    // constructor
    CartesianPoint(double xCoord, double yCoord) {
        x = xCoord; y = yCoord;
    }
    public double x() { return x; }
    public double y() { return y; }
    public Point plus(Point p) {
        return new CartesianPoint(x+p.x(), y+p.y());
    }
    public boolean greaterThan(Point p) {
        return (x>p.x()) & (y>p.y());
    }
    ...
}
```
Why is method request slow?

1. String compare
2. Linear Search
3. Chaining through super dictionaries

Why does it matter?

It doesn’t matter

- So long as there is a virtual machine interpreting the byte-code instructions, the overhead of method request is not much of a problem
How to speed-up OO?

- Compile them!
- Translate each byte code into the equivalent series of machine instructions
  - the very same instructions that the interpreter would have executed
- `method Request` is now a subroutine
  ... and it’s time-consuming
  Recall why:

String Compare

- String comparison is slow (linear in the length of the shorter string)
  - Avoid by using the Flyweight Pattern
    - see Smalltalk class Symbol

Linear Search

- Linear Search is slow
  - Linear in the number of methods
- Avoid by hashing
  - hash can be generated at compile time
    - hash function should be part of the language!
  - Hashing is constant time, provided _____________
- Space is not free

Why is this slow?

- Chaining through super dictionaries
  - Avoid by copying down super methods at compile time
  - e.g., Point inherits Object->printString, so copy the pair ⟨ #printString, code ptr ⟩ into Point’s method dictionary.
- Two problems:
  1. super-sends
  2. space consumption
**Simple Cache**

- Small cache indexed by pair \( \langle \text{receiver class, method name} \rangle \)
- Speeds up overall system by 20% to 30% [Krasner 1983], 37% [Hölzle 1981]
- But: there are lots of classes in the system!

**Per request-site Cache**

- Idea: use a separate cache for each method request site. [Deutsch POPL 1983]: Efficient Implementation of Smalltalk
- Locality says that most of the receivers at a given site will be of the same class
- e.g., list.collect { each \( \rightarrow \) each.display }
- if list is homogeneous, all of the convert requests will be to the same method
- Also: method name is now a constant

**How to find the Cache?**

- if you use one cache for each method request in the program, there will be a lot of caches
- make caches small, e.g., one entry!
- How do we find the right cache?
- Simple and effective solution: place the cache “in-line”: in the code in place of the original request!

(3@4) display

![Inline Caching Diagram](image)

Figure 1. Inline Caching

**Inline Caching**

- Exploits locality of call site
- Site is originally “unlinked”:
  - Jumps to the general lookup routine
- After first request, site is over-written with call to the “prologue” of the found method
  - Prologue checks that the class of the receiver is that expected by the method
  - If it’s not, jump to general lookup routine

**Inline Caching is Effective**

- 95% effective for Smalltalk
- Overall speedup of 50% on SOAR
- Can be combined with simple \( \langle \text{receiver class, method name} \rangle \) cache to handle misses.

**What about Polymorphic Sends?**

- Example: \( \text{array := \#(1 'a' 2 'b' 3 'c' 4 'd' 5 'e')} \)
  \( \text{array do: \[ :\text{each} | \text{each printOn: Transcript} \]} \)

- Worst case for inline-cache:
  - Why?

**Polymorphic Sends**

- Degree of Polymorphism is usually small
  - Less than 10
- If it’s not small, then it’s large
  - Trimodal distribution: monomorphic, polymorphic, megamorphic.
Polymorphic Inline Caches

- Suppose that we are displaying the elements of a list.
  - So far, every element has been a Rectangle.

- Now suppose that the next element is a circle.

- Suppose the next object is a Triangle.
  - PIC stub routine misses, but is extended with a third case:
    - PIC now handles Rectangles, Circles and Triangles.
  - Eventually, the PIC will handle all cases seen in practice.
  - If the size of the PIC grows too large:
    - Mark request site as megamorphic and quit caching.
Why Inline Caches Win

- They replace indirect calls by direct calls
- Modern hardware optimizes direct calls, e.g., with pipelining and lookahead
- The direct call is “right” 95% of the time

Variations

- Inline small methods into PIC stub
- Order classes in PIC by frequency
- Replace linear search by hashing, binary search, etc.
- Sharing PICS between request sites that have same method name
  - saves space, looses locality

Another Approach

- Use indirect calls
  - Compile the method name to a small integer that is used as a table index
- Every class has it’s
  - x method at offset 0, its
  - y method at offset 1, its
  - printOn method at offset 2, etc.

PICs first Implemented for Self

- Parser
- PrimMaker
- UI
- PathCache
- Richards
- PolyTest

Execution times relative to Self system with inline cache

PolyTest. An artificial benchmark (20 lines) designed to show the highest possible speedup with PICs. PolyTest consists of a loop containing a polymorphic send of degree 5; the send is executed a million times. Normal inline caches have a 100% miss rate in this benchmark (no two consecutive sends have the same receiver type).

Execution cache miss ratios

Figure 5. Impact of PICs on performance

Figure 6. Inline cache miss ratios
VTable for Virtual methods

Point object

<table>
<thead>
<tr>
<th>vptr</th>
<th>x</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>y</td>
<td>3</td>
</tr>
</tbody>
</table>

g getX
...translate
code
code
code
code

ColorPoint object

<table>
<thead>
<tr>
<th>vptr</th>
<th>x</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>y</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>color</td>
<td>red</td>
</tr>
</tbody>
</table>

g getX
...translate
getColor
code

code

code

code

code

code

vTables

- use multiple indirection instead of search
- hard to do with multiple inheritance
- a great source of research papers
- loose on modern architectures
  - no branch prediction through indirect call

AbCon Vectors